# МИНИCTEPCTBO НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ

**ФЕДЕРАЦИИ**

**Федеральное государственное автономное образовательное учреждение высшего образования**

# «СЕВЕРОКАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

**Кафедра инфокоммуникаций Институт цифрового развития**

# ОТЧЁТ

## по лабораторной работе

Дисциплина: «Объектно – ориентированное программирование»

|  |
| --- |
| Выполнил студент группы  ИВТ-б-о-21-1 |
| Богдашов А.В. « » 20 г. |
| Подпись студента |
| Работа защищена « » 20 г. |
| Проверил доцент  Кафедры инфокоммуникаций, старший преподаватель  Воронкин Р.А.  (подпись) |

Ставрополь 2023

## Классы данных в Python

**Цель работы:** приобретение навыков по работе с классами данных при написании программ с помощью языка программирования Python версии 3.x.

## Ход работа:

**Индивидуальное задание.**

Выполнить индивидуальное задание лабораторной работы 4.5, использовав классы данных, а также загрузку и сохранение данных в формат XML.

Код программы:

#!/usr/bin/env python3  
# -\*- coding: utf-8 -\*-  
  
from dataclasses import dataclass  
import xml.etree.ElementTree as ET  
  
  
@dataclass  
class Money:  
 rubles: int = 0  
 kopecks: int = 0  
  
 def read(self) -> None:  
 self.rubles, self.kopecks = map(int, input("Введите количество рублей и копеек через пробел: ").split())  
  
 def display(self) -> None:  
 print(f"{self.rubles} руб. {self.kopecks:02d} коп.")  
  
 def add(self, other: 'Money') -> 'Money':  
 total\_kopecks = self.rubles \* 100 + self.kopecks + other.rubles \* 100 + other.kopecks  
 return Money(\*divmod(total\_kopecks, 100))  
  
 def subtract(self, other: 'Money') -> 'Money':  
 total\_kopecks = self.rubles \* 100 + self.kopecks - (other.rubles \* 100 + other.kopecks)  
 return Money(\*divmod(total\_kopecks, 100))  
  
 def divide\_sum(self, num: float) -> 'Money':  
 total\_kopecks = (self.rubles \* 100 + self.kopecks) / num  
 return Money(\*divmod(int(total\_kopecks), 100))  
  
 def multiply\_by\_number(self, num: float) -> 'Money':  
 total\_kopecks = (self.rubles \* 100 + self.kopecks) \* num  
 return Money(\*divmod(int(total\_kopecks), 100))  
  
 def compare(self, other: 'Money') -> bool:  
 return self.rubles == other.rubles and self.kopecks == other.kopecks  
  
 def is\_less\_than(self, other: 'Money') -> bool:  
 return (self.rubles \* 100 + self.kopecks) < (other.rubles \* 100 + other.kopecks)  
  
 def to\_xml\_element(self) -> ET.Element:  
 money\_element = ET.Element('Money')  
 ET.SubElement(money\_element, 'Rubles').text = str(self.rubles)  
 ET.SubElement(money\_element, 'Kopecks').text = str(self.kopecks)  
 return money\_element  
  
 @classmethod  
 def from\_xml\_element(cls, element: ET.Element) -> 'Money':  
 rubles = int(element.find('Rubles').text)  
 kopecks = int(element.find('Kopecks').text)  
 return cls(rubles, kopecks)  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 money1 = Money()  
 money1.read()  
 money1.display()  
  
 money2 = Money()  
 money2.read()  
 money2.display()  
  
 sum\_result = money1.add(money2)  
 print("Сумма:")  
 sum\_result.display()  
  
 diff\_result = money1.subtract(money2)  
 print("Разность:")  
 diff\_result.display()  
  
 divide\_sum\_num = float(input("Введите число для деления суммы: "))  
 div\_sum\_result = money1.divide\_sum(divide\_sum\_num)  
 print("Деление суммы на число:")  
 div\_sum\_result.display()  
  
 multiply\_by\_num = float(input("Введите число для умножения: "))  
 mul\_result = money1.multiply\_by\_number(multiply\_by\_num)  
 print("Умножение на число:")  
 mul\_result.display()  
  
 comparison\_result = money1.compare(money2)  
 print(f"Сравнение: {comparison\_result}")  
  
 comparison\_result\_lt = money1.is\_less\_than(money2)  
 print(f"Сравнение меньше: {comparison\_result\_lt}")  
  
 # Сохранение в XML  
 money1\_element = money1.to\_xml\_element()  
 tree = ET.ElementTree(money1\_element)  
 tree.write('money\_data.xml')  
  
 # Загрузка из XML  
 loaded\_tree = ET.parse('money\_data.xml')  
 loaded\_money\_element = loaded\_tree.getroot()  
 loaded\_money = Money.from\_xml\_element(loaded\_money\_element)  
 print("Данные, загруженные из XML:")  
 loaded\_money.display()

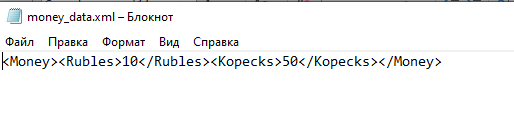


Рисунок1. Результат выполнения

## Контрольные вопросы:

1. **Как создать класс данных в языке Python?**

В Python создание класса данных осуществляется с использованием ключевого слова class. Вот пример простого класса данных:

class Person:

def init (self, name, age): self.name = name

self.age = age

# Создание экземпляра класса person1 = Person("Иван", 25)

# Доступ к атрибутам экземпляра класса print(person1.name) # Выведет: Иван print(person1.age) # Выведет: 25

В этом примере мы создаем класс Person, который имеет атрибуты name и age. Метод init является конструктором класса и используется для инициализации атрибутов при создании экземпляра класса. При создании экземпляра класса Person мы передаем значения для атрибутов name и age.

Доступ к атрибутам экземпляра класса осуществляется с использованием точки (например, person1.name).

Это только простейший пример класса данных. В Python классы могут содержать методы (функции, связанные с классом), наследование, статические методы, свойства и многое другое.

## Какие методы по умолчанию реализует класс данных?

В Python класс данных может реализовывать несколько встроенных методов по умолчанию, которые позволяют определить специальное поведение объекта. Некоторые из этих методов включают:

1. init (self, ...): Конструктор класса, который вызывается при создании нового экземпляра класса.
2. str (self): Метод, который возвращает строковое представление объекта. Он вызывается, когда объект передается функции str() или когда объект используется в строковом контексте.
3. repr(self): Метод, который возвращает представление объекта, которое может быть использовано для его воссоздания. Он вызывается, когда объект передается функции repr() или когда объект используется в интерактивной оболочке Python.
4. eq (self, other): Метод для сравнения объектов на равенство (используется оператор ==).
5. lt (self, other), le (self, other), gt (self, other), ge (self, other): Методы для сравнения объектов (используются операторы <, <=, >, >=).
6. hash (self): Метод для вычисления хэш-значения объекта, используемого в словарях и множествах.
7. getattr (self, name), setattr (self, name, value): Методы для перехвата доступа к атрибутам объекта.
8. del (self): Метод, который вызывается при удалении объекта.

Это только небольшой набор методов по умолчанию, которые могут быть реализованы в классе данных. В Python есть еще много других "магических" методов, которые позволяют определить специальное поведение объектов.

## Как создать неизменяемый класс данных?

В Python неизменяемый класс данных можно создать, используя неизменяемые типы данных в качестве атрибутов класса, и предоставляя только методы для чтения значений атрибутов, но не для их изменения. Вот пример создания неизменяемого класса данных:

class ImmutableData:

def init (self, value1, value2):

атрибут

self.\_value1 = value1 # Префикс "\_" обозначает "приватный"

self.\_value2 = value2 def get\_value1(self):

return self.\_value1

def get\_value2(self): return self.\_value2

В этом примере атрибуты value1 и value2 являются приватными (по соглашению обозначены префиксом \_), и доступ к ним осуществляется только через методы get\_value1 и get\_value2. Таким образом, значения атрибутов не могут быть изменены напрямую извне.

Пример использования:

data = ImmutableData(10, 20) print(data.get\_value1()) # Выведет: 10 print(data.get\_value2()) # Выведет: 20

# Попытка изменить значение атрибута вызовет ошибку data.\_value1 = 100 # AttributeError: can't set attribute

Этот подход позволяет создать неизменяемый класс данных, в котором значения атрибутов не могут быть изменены после создания экземпляра класса.

**Вывод:** в ходевыполнения данной лабораторной работы были приобретены навык по работе с классами данных при написании программ с помощью языка программирования Python версии 3.x.